Repères

JAVA et les AUTOMATISMES PROGRAMMABLES

Sujet: Java et les automatismes programmables
Verbe: Pister
Complément: Java, technologie banalisée par Internet est en passe de pénétrer le marché des automatismes programmables. Points faibles et points forts de Java

Java, technologie nouvelle banalisée par Internet via les célèbres applets chargées dans votre browser préféré (Navigator de Netscape, Explorer de Microsoft etc.), pénétrera peut-être le marché des automatismes programmables. Rappelons qu'un browser est un interface de navigation sur Internet alors qu'une applet est une petite application importée automatiquement à travers le réseau pour s'exécuter dans le browser.

Le *Network Computer* (NC) est un ordinateur sans disque dur utilisant un serveur sur le réseau comme mémoire de masse. Ce n'est pas de la science-fiction d'imaginer que l'ingénieur pourra demain utiliser un NC chez lui pour visualiser et régler le processus de production. Cet aspect sera repris dans cet article plus dans le détail.

Il convient cependant de garder la tête froide et de ne pas céder à l'effet de mode en positionnant Java sur tous les créneaux.

Le milieu industriel est assez conservateur en terme de technique informatique. Il existe en gros deux publics dans le monde des automatismes : l'ingénierie des grands comptes qui sert souvent de banc d'essais en terme de technologie nouvelle et les petits utilisateurs / installateurs qui acceptent facilement les innovations lorsqu'elles simplifient le travail.

Une chose paraît acquise, le phénomène Java a atteint une taille critique. Les intérêts économiques et stratégiques en jeu laissent penser que le phénomène va s'amplifier.

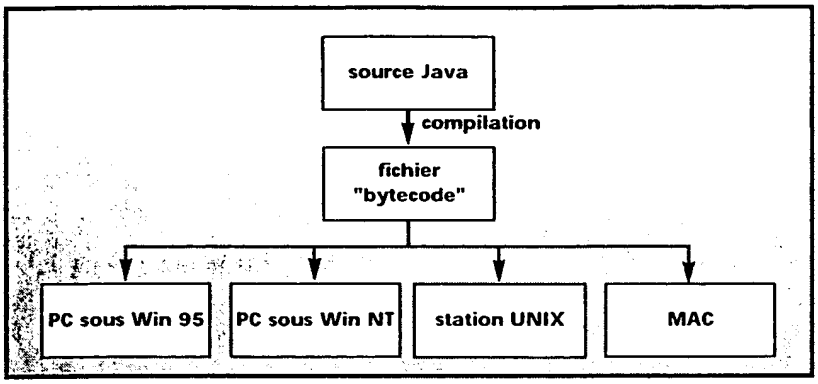
Cet article indique des voies à explorer en terme de faisabilité technique ainsi que du point de vue marketing. **Qu'est-ce que Java peut apporter à l'utilisateur** ? Solutions moins onéreuses, multi fournisseurs, multi plates-formes, ouverture de l'atelier logiciel ?

La suite de ce papier aborde les sujets suivants :
- les apports techniques de Java,
- les points faibles de cette technologie,
- une vision de l'usine intégrée à l'intranet entreprise,
- les impacts potentiels côté atelier logiciel et côté application embarquée.

PRINCIPALES CARACTÉRISTIQUES DE JAVA

Ce langage simple a été développé par Sun. Il est réellement **orienté objet** avec une syntaxe héritée de C/C++, une gestion mémoire propre *(garbage collector)* et le multitâches intégré *(thread)*. L'unité de compilation est la classe d'objet avec **une édition de liens réalisée à l'exécution** *(class loader)*.

Ce langage est **neutre**, c'est à dire qu'il peut s'exécuter dans n'importe quel environnement pourvu que son exécutif



soit intégré dans l'environnement, ce qui est le cas dans certains *browsers*. Cet exécutif, la *Virtual Machine* (VM), est un processeur logiciel. La neutralité est dûe à l'utilisation d'un code intermédiaire, le *bytecode*, exécuté par la VM. C'est la version *bytecode* d'une *applet* qui est chargée à travers le réseau.

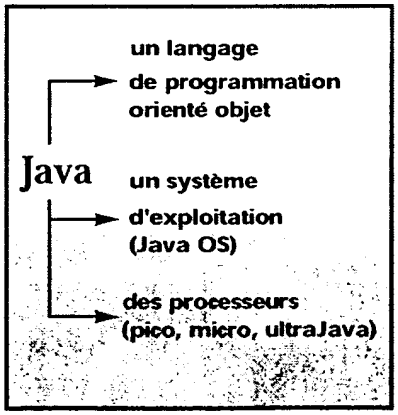
Java est très orienté **distribution / coopération sur les réseaux** via l'intégration de bon nombre de protocoles dans l'environnement de programmation (TCP/IP, HTTP, FTP etc.). Java peut aussi faire interagir des objets sur le réseau via le *Remote Method Invocation* (RMI). Le RMI, c'est la possibilité pour un objet d'exécuter une méthode d'un objet distant de façon quasiment transparente au niveau applicatif.

Un *browser* Java est **flexible**. Il est capable, pour dialoguer avec un serveur dont il ne connaît pas un protocole application, de charger en premier le driver du protocole avant de dialoguer avec le serveur. De même, il est capable, pour interpréter un format de données inconnu (un fichier graphique par exemple), de charger le driver d'interprétation du type de données avant d'afficher l'image associée au fichier.

Un NC Java est " client-centric ", c'est à dire que **l'application est exécutée par le NC**. En conséquence, le NC Java est capable de se charger à partir d'un Système d'Exploitation (SE) minimal, de récupérer le reste de son SE, un *browser* et les applications dont il a besoin à partir d'un serveur sur le réseau. Cette ouverture sur les réseaux s'accompagne d'une politique de sécurité grâce entre autres au *bytecode* verifier qui contrôle la cohérence d'une *applet* avant de l'exécuter.

L'environnement Java est pourvu d'un grand nombre de bibliothèques de classes structurées en API *(Application Programming Interface)* de base (*core* API) et extension (*extension* API). Citons par exemple la " Java Enterprise API " qui permet de s'interfacer à un serveur SQL et au monde CORBA *(Common Object Request Broker Architecture)*.

Il faut aussi mentionner le modèle de composant logiciel identifié sous la dénomination **Java Beans** (*Bean* pour grain de café). Un *Bean* est un composant logiciel réutilisable qui peut être manipulé visuellement dans un outil de construction d'application. Son rôle est comparable à celui des contrôles ActiveX de Microsoft. L'interface d'un *Bean* est standardisé, il est facile de les faire coopérer.



Il est possible **d'interfacer une application Java avec un autre langage** (C, C++, assembleur etc.) via la *native interface* pour diverses raisons : performances, utilisation d'une bibliothèque graphique existante ... Attention, c'est la porte ouverte à la non-portabilité d'une application Java.

Java, **c'est aussi un SE complet**, le JavaOS de Sun, qui permet de supporter une application Java sur un processeur quelconque avec un minimum de ressources. C'est le concept de *thin client* que l'on retrouve dans le NC. La cible n'a plus à supporter un encombrant Windows NT ou UNIX si toutes les applications sont écrites en Java.

Ajoutons enfin les **processeurs Java** (pico, micro et ultraJava annoncés par Sun) qui exécuteront directement le *bytecode* dans un environnement JavaOS. Cette gamme de processeurs couvrira les besoins de l'embarqué avec peu de ressources jusqu'à la station de travail. Sun sort le picoJava1 en technologie RISK à 100 Mhz.

Points faibles de Java

Java a été lancé il y a deux ans. Malgré son développement très rapide, il manque de maturité, notamment au niveau de ses API dont certaines sont en cours de spécification. Il y a **risque de divergence comme pour le langage C ou UNIX** avec plusieurs standards en fin de course, donc des problèmes de compatibilité. En particulier, l'existence chez Netscape de l'*Internet Foundation Classes* (IFC) et chez Microsoft de l'*Application Foundation Classes* (AFC), déclinaisons de l'API graphique Java *Abstract Window Toolkit* (AWT), illustre la difficulté de converger. Cependant, Sun met en place le label " 100% pure java " afin de certifier les applications qui respectent le vrai standard.

Java est interprété dans les *browsers* avec des **performances ne correspondant pas forcément au besoin de certaines applications**. La technique JIT *(Just In Time compiler)* de compilation à la volée permet d'améliorer les choses au prix d'une expansion non négligeable du volume de code. Les puces Java augmentent les perfor-

mances par rapport à la compilation. Nous sommes en droit d'espérer que le code Java atteindra les performances du C/C++ au moins dans la version silicium.

La taille mémoire utilisée par une application Java et son environnement n'est pas négligeable (de l'ordre de plusieurs Mo). Ce n'est pas grave pour un PC, ça l'est plus pour l'embarqué : la définition de profils spécifiques pour l'embarqué est une nécessité, même en utilisant le JavaOS.

Le portage d'applications de C++ vers Java indique que la consommation mémoire augmente très sensiblement (voir l'article de 01 Informatique du 14 mars 97: " les défauts de jeunesse de Java : les performances et la portabilité ").

Le sujet le plus délicat dans notre domaine d'application est **le caractère non " temps réel dur " de Java**. K. Nielsen, université de l'Iowa et président de la société NewMonics (http://www.newmonics.com), a identifié ces points faibles. Sans entrer dans le détail, en voici la liste : le *garbage collector* (allocation / libération de la mémoire), l'ordonnancement des tâches, la synchronisation des tâches (risque d'inversion de priorité etc.), le manque de moyen d'analyse de l'exécution d'une application. Il propose une API qui est censé résoudre ces problèmes. Est-ce le cas ? Y-a-t-il d'autres voies ? Le chemin est certainement encore long avant de pouvoir programmer temps réel en Java dans un environnement standardisé.

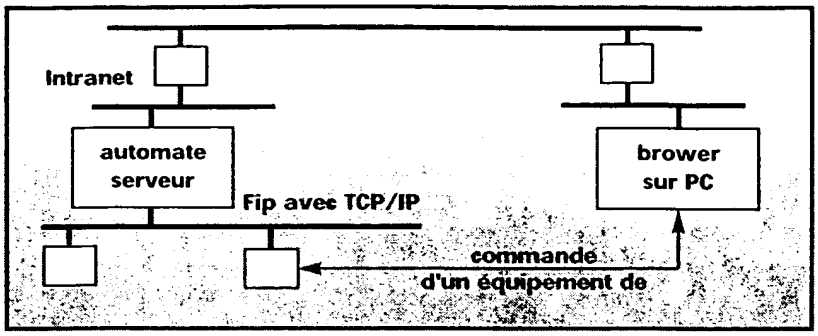
La suite de cet article propose des applications de la technologie Java pour les automatismes programmables en supposant que les difficultés listées ci-dessus sont résolues (tout ou partie), ce qui est loin d'être acquis.

L'INTRANET, CANAL FÉDÉRATEUR DES FLUX D'INFORMATION DANS L'ENTREPRISE

Il est probable que **TCP/IP sur Ethernet devienne le réseau fédérateur de l'usine**, de toute l'entreprise, via l'Intranet. Examinons quel serait son nouveau positionnement en production.

L'application d'automatismes est répartie sur les réseaux de niveau 0 (capteurs/actionneurs : ASI, Seriplex etc.), niveau 1 (périphérie de l'automatisme : Fip, Profibus etc.) et niveau 3 (supervision : TCP/IP sur Ethernet ou autres).

Si le niveau 0 n'a généralement pas la capacité à supporter TCP/IP, il en est autrement du niveau 1. **La cohabitation de TCP/IP sur le niveau 1** avec



les protocoles existants (MPS, FMS etc.) a de bonnes chances de se concrétiser. N'oublions pas que ce type de réseau doit respecter des contraintes de tenue CEM, de déterminisme de la couche liaison et de temps de réponse de l'équipement à une requête.

L'utilisation de *browsers* va se généraliser en synergie avec un certain nombre de serveurs. L'automatisme programmable se doit de **supporter entre autres TCP/IP pour réussir son intégration dans l'Intranet**. Nous verrons certainement dans les années à venir un nombre croissant d'équipements connectés à l'Intranet. Les problèmes de sécurité sont différents selon que l'utilisateur connecté à l'automatisme se trouve sur l'Intranet (sécurité allégée) ou sur Internet (sécurité plus stricte, notamment via la technique du *firewall*).

Les impacts potentiels coté atelier logiciel

Le terme atelier logiciel recouvre l'atelier de développement, la console de réglage / diagnostic, l'équipement de supervision...

Avant tout, Java est un langage de programmation intrinsèquement plus productif que C/C++ (gestion mémoire propre, maquettage etc.). C'est une raison suffisante pour l'utiliser comme outil de développement de l'atelier logiciel : **le produit sera moins cher !**

Comme Java est neutre (sauf problème de portabilité en cas d'usage de la *native interface*), **l'atelier logiciel est de base multi plates-formes.** Même si le parc installé est surtout à base de PC sous Windows95 / NT, les autres environnements deviennent utilisables. Si les utilisateurs nous demandent de faire fonctionner tout ou partie de l'atelier logiciel sur un NC Java, l'opération sera immédiate dans la mesure ou l'atelier est déjà codé en Java. Cet atelier Java **bénéficiera de la technologie *Bean*** (modularité, ouverture) et d'un environnement de pointe tiré par les besoins de la sphère Intranet / Internet. Il est difficile aujourd'hui de présager du succès des *Beans* : il se mesurera à la richesse des bibliothèques de composants.

Reprenons l'exemple de l'automaticien équipé d'un NC connecté à l'Intranet et désireux de régler une variable du processus via un automate programmable. Le serveur ayant servi à charger le NC pourrait se trouver dans l'automate (avec la mémoire de masse suffisante) ou bien être localisé sur l'Internet. Dans ce dernier cas, **la connexion à un site du fournisseur servirait à récupérer la derrière version de l'application** qu'il doit utiliser. Ce genre d'opération est réalisable grâce à la flexibilité du *browser* Java.

Abordons maintenant le domaine de **la génération de code embarqué** (qu'il s'exécute sur un automate de type PC ou conventionnel) en supposant les problèmes temps réels et volume mémoire résolus. Java apporte la modularité, la répartition et un compilateur. Nous pouvons imaginer qu'un bloc IEC1131-3, un réseau ladder, un grafcet, sera généré en code Java avec la classe comme granularité. Deux options se dessinent. Premièrement, la cible d'exécution n'a pas de caractéristiques temps réel dur : le modèle de compilation par classe avec édition de liens à l'exécution s'applique. Deuxièmement, la cible exécute du code dans un environnement plus contraint : si la compilation reste modulaire, il est préférable de faire l'édition de liens dans la foulée, le résultat étant directement exécutable par la cible.

Pour les utilisateurs informaticiens, il serait envisageable d'offrir la possibilité de programmer tout ou partie de l'application directement en Java, de se fabriquer ses Beans. L'utilisation d'un langage de script Java, plus simple que Java, n'est pas à écarter.

Le ***Bean*** pourrait servir à structurer l'application, à en faciliter l'écriture (composition d'objets graphiques) et surtout à **faire de la réutilisation**. Il n'y aurait plus de programmation classique, mais de l'assemblage et du paramétrage de composants, donc **réduction du coût de l'automatisme**. Qui plus est, le fournisseur d'un équipement pourrait livrer le *Bean* approprié pour insérer harmonieusement cet équipement dans l'application d'automatisme. Tout ceci passera par de gros efforts de standardisation et par la création d'un marché de composants logiciels pour l'embarqué.

Les impacts potentiels coté exécution de l'application d'automatisme

Le domaine couvert correspond à l'automate en rack, l'automate PC et tous les équipements participants à l'automatisme. Il n'est pas interdit d'étendre le champ aux machines spécialisées comme les commandes numériques, robots...

L'application la plus immédiate est le " **frontal Java** ", c'est à dire un serveur Java implémenté dans un coupleur de communication TCP/IP qui servirait de passerelle vers l'automate afin de pouvoir au minimum régler les variables de l'application. C'est facilement réalisable grâce à la technique de *native interface*. Une expérience similaire a été réalisée sur un automate Quantum : implémentation d'un petit serveur Web sans passerelle vers l'automate. Elle est en cours d'évaluation sur le Premium.

Exécuter du code Java faciliterait la **coopération** au sein de l'application d'automatisme distribuée **grâce au**

Remote Method Invocation (RMI). C'est également l'ouverture vers le monde CORBA, vers les serveurs de base de données (SQL ou autres). L'utilisation de serveurs dans les automatismes sera de plus en plus importante avec la montée en puissance de l'Intranet. Si Java est adopté par les grands fournisseurs d'automatismes, la mixité des installations posera moins de problèmes.

Pour exécuter du code Java, il faut **implémenter la *Virtual Machine*** (VM) de préférence dans un système d'exploitation dédié comme JavaOS. Pour les performances, **utiliser une puce Java** sera certainement un passage obligé. Mettre la VM dans un ASIC ne semble pas être hors de portée, gageons que nous verrons un jour sur le marché des microprocesseurs classiques supportant en supplément le *bytecode*.

Le coût de l'environnement d'exécution Java est le critère permettant de positionner la barrière entre Java et la technologie actuelle dans l'embarqué. **Plus le coût de l'équipement sera élevé, plus il y aura de chances d'utiliser Java** : par exemple, une E/S analogique n'entrera pas dans ce cas de figure.

Il se dessine au moins trois tendances côté exécutif Java. La première est **le portage de la VM dans un SE temps réel** comme Vxworks de WindRiver ou Psos de ISI. Dans le cas de Vxworks, la VM est exécutée dans une tâche peu prioritaire, les tâches plus prioritaires s'exécutent de façon temps réel. Dans cette solution, le code Java n'est pas temps réel. La seconde consiste **à utiliser JavaOS en modulant le nombre d'API en fonction des besoins** (par exemple pas de fenêtrage). Sur le site JavaSoft (http://www.javasoft.com), il est fait référence à une " embedded API " qui devra définir un sous-ensemble des API de base pour l'embarqué. Cette solution aurait le mérite d'être standard. Une troisième option, consiste **à porter la VM minimale dans un environnement propriétaire**. Une telle expérience a été réalisée à l'université de Californie, Santa Cruz, avec le " Java Nanokernel " (JN) (http://www.csc.ucsc.edu/research/embedded/pubs/tr96-29.html). Bien que l'aspect temps réel n'ait pas été pris en compte, cette démarche semble prometteuse pour de petits environnements.

En conclusion

Si la pénétration de Java dans le développement des applicatifs de l'atelier logiciel ne doit pas poser de gros problèmes, il en est autrement pour le logiciel embarqué. En octobre 96, Sun a annoncé la création d'un groupe de travail sur la **"Java Automation API"** en collaboration avec des entreprises du métier manufacturier comme ABB Systems Control, The Foxboro Company, Hewlet-Packard Company, SAP, Toshiba Company ... L'objectif est de permettre le développement d'applications Java indépendantes de la plateforme, assurant le contrôle de procédé temps réel et prenant en compte la dimension Intranet et Internet. Les spécifications sont attendues pour mi-97.

Si la technologie Java permet de **diminuer les coûts d'installation et de maintenance** des automatismes ou si elle facilite le **support de nouvelles fonctionnalités** comme la connexion de l'automatisme à l'Intranet, les fournisseurs ne devraient pas tarder à utiliser Java. ■

Gérard Picon
Schneider Automation
Service Logiciel Interface et Application